Asynchronous MMC based Parallel SA Schemes for Multiobjective Standard Cell Placement

Sadiq M. Sait Ali Mustafa Zaidi Mustafa Imran Ali

College of Computer Sciences & Engineering
King Fahd University of Petroleum & Minerals
Dhahran 31261, Saudi Arabia
E-mail: {sadiq, alizaidi, mustafa}@ccse.kfupm.edu.sa

***Abstract*— Simulated Annealing (SA) is a popular iterative heuristic used to solve a wide variety of combinatorial optimization problems. However, depending on the size of the problem, it may have large run-time requirements. One practical approach to speed up its execution is to parallelize it. In this paper we develop parallel SA schemes based on the Asynchronous Multiple-Markov Chain model (AMMC) described in [1] and applied to standard-cell placement in [2]. The schemes are applied to solve the multi-objective standard cell placement problem using an inexpensive cluster-of-workstations environment. This problem requires the optimization of conflicting objectives (interconnect wire-length, power dissipation, and timing performance), and Fuzzy logic is used to integrate the costs of these objectives [3], [4]. Experiments are performed on ISCAS-85/89 benchmark circuits. Our goal is to develop parallel SA schemes that provide significantly improved runtime/solution quality characteristics for this key CAD problem, by making the best possible use of an inexpensive parallel environment.**

I. Introduction

With VLSI technologies moving towards ever more complex submicron-scale circuit fabrication, there is a demanding need for iterative heuristics that can deliver quality solutions in feasible runtimes. This is especially true with the often conflicting, multiple objectives that have be addressed. Stochastic Heuristics such as Simulated Annealing are very useful for obtaining near optimal solutions for this class of problems. However, there is the pressing issue of huge runtime requirements that can grow exponentially with circuit size.

One way to adapt iterative techniques to solve large problems and traverse larger search spaces in reasonable time is to parallelize them [5], [6]. The eventual goal being to achieve either much lower run-times for same quality solutions, or higher quality solutions in a fixed amount of time. This however is easier said than done: an effective parallelization strategy must consider issues such as proper partitioning of the problem to facilitate uniform distribution of computationally intensive tasks, and enabling a more thorough traversal of the complex search space, all the while respecting the benefits and limitations of the selected parallel environment.

Parallel Simulated Annealing has been the subject of thorough exploration since it was first proposed. Virtually all known methods of parallelization for Simulated Annealing can be classified into one of two groups: Single Markov-chain and Multiple Markov-chain methods [1]. Most Single Markov chain approaches attempt to exploit parallelism between the three phases. They include move-acceleration, parallel-moves, and speculative annealing. These techniques are generally more suitable for shared-memory environments. Approaches based on Multiple Markov chains call for the concurrent execution of separate simulated annealing chains with periodic exchange of solutions [1], [2], and are ideally suited for distributed memory systems, considering that the need for communication between nodes is considerably reduced.

In our work, we experiment with different versions of the Asynchronous Multiple-Markov Chain Parallel SA (or AMMC PSA) approach described in [1], as this scheme is most suitable for a basic VLSI cell placement problem being solved using a cluster-of-workstations environment [2].

II. Parallelization Schemes and Results

A. Experimental Setup and Placement Cost-Functions

The experimental setup consists of a homogenous cluster of 8 Pentium-4 2 GHz machines, and 256 MB of memory. These machines are connected by 1Gbit/s ethernet switch. Operating system used is Redhat Linux 7.3 (kernel 2.4.7-10). The algorithms were implemented in C/C++, using MPICH ver. 1.2.4. ISCAS-89 circuits are used as performance benchmarks for evaluating the parallel metaheuristics.

Our placement optimization problem is of a multiobjective nature with three design objectives namely, interconnect wire-length, power consumption, and timing performance (delay). The layout width is taken as a constraint. Fuzzy logic was used for designing an aggregating cost function, allowing us to describe the objectives in terms of linguistic variables. Then, fuzzy rules are used to find the overall cost, or 'quality' of a placement solution. This quality measure is a value between 0 and 1, with 1 representing an optimal solution. A detailed description of our combinatorial optimization problem and cost functions is given in [3].

B. Attempted Parallelization Strategies

Recent work has shown that the most promising scheme for the parallelization of SA for the VLSI Placement problem in a cluster-of-workstations environment was the Asynchronous MMC model [1], [2]. The primary goals of our experiments were to explore the potential for improvements in runtime

and achievable solution quality, by making the most effective utilization of the parallel environment. Successive parallel strategies attempt to incrementally build upon the knowledge gathered from the previous schemes in order to achieve the goals of parallelization more effectively.

The basic structure of our AMMC PSA implementation is similar to the scheme described in [2]. On each available processing element, an SA operation is initiated with the same starting solution, but with different seeds for pseudo-randomization. The specifications of our AMMC parallel search implementation of SA are given below:

1) The Information Exchanged: The entire recent best solution is communicated to slave processes.
2) Connection Topology: The parallel processes communicate via a central solution storage area, where the best solution found so far is kept. The master process is reserved for this purpose.
3) Communication Mode: Communication is asynchronous. Thus communication time is minimized since there are no synchronization barriers. Each process communicates with the master independently and compares its own best solution with the solution residing at the master. If the master owns the better solution, the slave starts its next Metropolis loop with this solution, while the master's copy remains unchanged. Conversely, if the slave has the better solution, it continues its work after the master has received this latest best solution, which is then available for comparison by the other slave processes.
4) Time to Exchange Information: Each process works on a recent best solution retrieved from the central store for the duration of its Metropolis loop.

We implement four distinct versions of the Asynchronous Multiple Markov Chains approach.

1) Asynchronous MMC Parallel SA Strategy 1: For Strategy 1, aside from the above points, there is no difference between the serial version and each of the parallel search processes. Such an approach has been found to improve solution qualities in a fixed amount of time [1], and our results corroborate this fact.

TABLE I

RESULTS FOR STRATEGY 1

Circuit Name	Number of Cells	$\mu(s)$ SA	Serial SA Time	Time for Parallel SA Strategy 1				
				p=3	p=4	p=5	p=6	p=7
s1196	561	0.675340	190	145.98	130.95	110.31	96.98	98.24
s1238	540	0.699469	212	183.91	130.32	127.55	117.12	114.66
s1488	667	0.650381	275	151.46	118.44	112.59	98.94	94.04
s1494	661	0.647920	214	131.40	116.27	101.89	98.13	92.26

Table I shows the results obtained from experiments with Strategy 1 for the benchmark circuits listed in column 1. The third column lists the highest quality achieved by the serial version of the algorithm. The remaining columns list the time taken to achieve the specified quality, with the given number of processors. Using Strategy 1, we were always able to exceed the quality achieved by the serial version. Figure 1 (a) shows the speedups achieved by Strategy 1, for the same quality, with different number of processors and for different circuits. Here we see that speedup achieved using Strategy 1 is sub-linear. Even with 8 processors, we are unable to even achieve a speedup of 3.

2) Asynchronous MMC Parallel SA Strategy 2: While Strategy 1 is able to meet and even surpass the qualities achieved by the serial algorithm, its runtime characteristics leave something to be desired. Strategy 2 is an attempt to provide near linear speedup over the serial version. This is accomplished by dividing the amount of work done at each of the individual processes by the total number of processes. Specifically, the number of Metropolis iterations at each process is divided by the total number of processes.

Table II shows the results obtained from experiments with Strategy 2. Unlike the previous table, the third column here shows the highest common quality that could be achieved by multiple runs of Strategy 2 for every number of processors. Comparing with column 3 of Table I, we can easily note that there is a roughly 10 to 15% drop in achievable quality with this scheme. Figure 1 (b) shows the speedups achieved by Strategy 2 as the number of processors is varied. We see that in this case, speedup is almost linear.

TABLE II

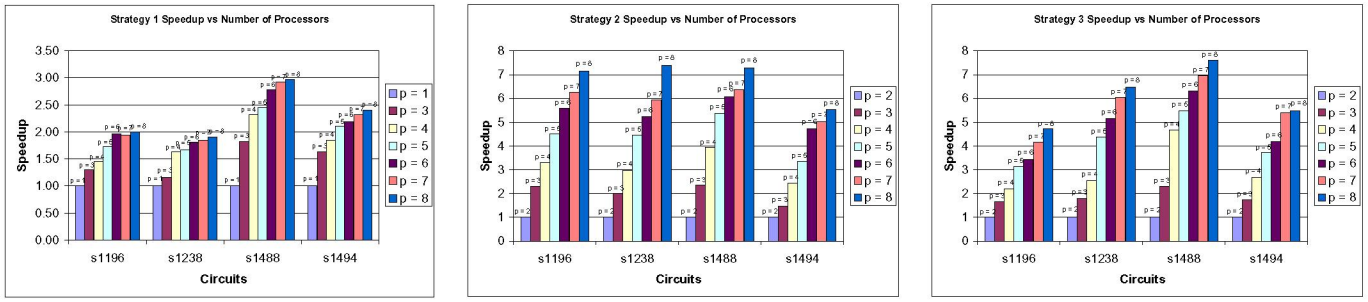
RESULTS FOR STRATEGY 2

Circuit Name	Number of Cells	$\mu(s)$ SA	Serial SA Time	Time for Parallel SA Strategy 2				
				p=3	p=4	p=5	p=6	p=7
s1196	561	0.630367	103	44.67	31.32	22.81	18.47	16.46
s1238	540	0.630573	117	58.03	39.21	26.31	22.31	19.73
s1488	667	0.582884	101	42.67	25.59	18.77	16.61	15.85
s1494	661	0.591114	75	51.11	30.79	22.32	15.82	14.9

3) Asynchronous MMC Parallel SA Strategy 3: The loss in achievable quality in Strategy 2 can be understood by looking at how the intelligence of the algorithm is affected by division of the factor 'M'. All of the parameters of the cooling schedule were originally optimized for the serial Simulated Annealing. Since SA convergence is highly sensitive to the cooling schedule, it is understandable that such a drastic change to one of its parameters would result in lower quality solutions. Division of 'M' reduces the amount of time each processor spends searching for a better solution in the vicinity of a previous good solution, resulting in a less thorough parallel search of the neighboring solution space.

In Strategy 3, we attempted to offset the negative impact on algorithmic intelligence by introducing other enhancements to the parallel algorithm. This was done by implementing different cooling schedules on each processor in such a way that some of the processors are searching for new solutions in a greedy manner, while others are still in the high temperature region. We essentially aim to counterbalance the impact of shortened Markov-chains on achievable quality by making intelligent use of the interaction between chains that occurs after every Metropolis loop.

This is different from the Temperature Parallel Simulated Annealing (TPSA) approach described in [7], which maintains all the parallel processes at constant but different temperatures. Whereas in Strategy 3, the values of *alpha* is different on different processors, thus the rate of temperature change is varied across processors. This is because our intended goals



(a) (b) (c)

Fig. 1. Speedup versus number of machines for (a) Parallel SA AMMC Strategy 1; (b) Parallel SA AMMC Strategy 2; (c) Parallel SA AMMC Strategy 3.

are different from those of TPSA. Whereas our primary aim is to achieve serial-equivalent qualities while achieving near-linear runtimes, the aim of TPSA was primarily to enhance the robustness of Parallel SA, and minimize the amount of effort required in parameter setting.

However, we find that even this proposed enhancement of varying *alpha* is insufficient to counteract the impact of divided 'M'. Our results for Strategy 3, shown in Table III and Figure 1(c) show no improvement over the results obtained for Strategy 2 - for some circuits (e.g. s1196), there is even a drop in achievable speedup and quality.

TABLE III
RESULTS FOR STRATEGY 3

Circuit Name	Number of Cells	$\mu(s)$ SA	Serial SA Time	Time for Parallel SA Strategy 2				
				p=3	p=4	p=5	p=6	p=7
s1196	561	0.606818	64	38.85	20.40	18.68	15.41	13.55
s1238	540	0.630573	117	65.36	26.65	22.65	19.39	18.04
s1488	667	0.582884	101	43.71	18.46	15.96	14.49	13.29
s1494	661	0.591114	75	42.89	20.05	17.92	13.86	13.67

4) Asynchronous MMC Parallel SA Strategy 4 - Adaptive Cooling Schedule: From the results of the previous three strategies, it became evident that for parallel SA, if any progress is to be made towards achieving our goals of near-linear run times with sustained quality, an in depth study of the impact of parameter M on achievable solution quality is required. To this end, we ran several experiments on both the serial and parallel (7 processors) versions, keeping all things constant except M, which was divided by 9, 17, 25, and 57 respectively for each new run. The Quality vs. Runtime results for the runs on 7 processors are given in Figure 2.

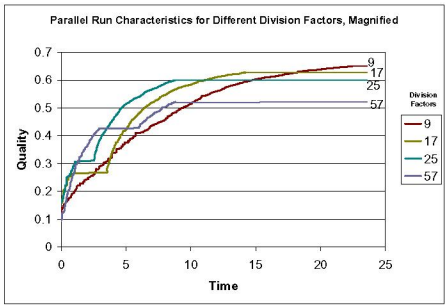


Fig. 2. Quality vs. Runtime results for AMMC Parallel SA (7 processors), with different values for M

From these results, we can see that division of M by a larger number increases the rate at which new solutions are found initially, but the system stagnates at a lower final solution quality. Intuitively this would suggest that the M factor should start at a small value, and then should increase as solution quality rises. However, a balance is necessary: if M increases too fast, runtime is compromised; if M increases too slowly, achievable solution quality is affected. The key to this dilemma of approximating the appropriate value of M comes from an interesting observation made during these runs: during the steep improvement phase the rate of improvements to solution quality is constant per metropolis call - meaning that during the initial phase, the high rate of climb is primarily due to the short time spent in each metropolis call.

Based on what we have learned from these experiments, we proposed certain modifications to the cooling schedule of our basic, serial Simulated Annealing algorithm. This adaptive cooling schedule, when implemented for the parallel AMMC scheme, yielded our 4th parallel search SA strategy. A brief description of the adaptive cooling schedule is given below:

1) For the first 100 or so Annealing iterations, accumulate an average of the quality improvement per Metropolis function call. This average rate of improvement will serve as a threshold that needs to be maintained per Metropolis Function call.
2) Initially, the value of 'M' is set to a very small value - the value used in the basic algorithm is divided by 25 to provide the initial M in the adaptive version.
3) After the initial average accumulation iterations, adaptivity is initiated. If rate of improvement drops below a certain threshold, increase M incrementally, since not enough time is being spent at each temperature level.
4) If rate of improvement is constantly more than the threshold value, decrease M, since an unnecessary amount of time is being spent at the given quality level.
5) The value of the M parameter is not allowed to exceed twice the value used in the original basic version, until significant stagnation is detected (e.g.: no improvement in solution quality for the past 25 Metropolis calls).

The application of the last condition was empirically found to dramatically improve algorithm run times, without sacrificing final quality achieved.

The run times for Serial and Parallel versions of Simulated Annealing with the adaptive cooling schedule are given in Table IV for the solution qualities achieved by Strategy 1. Table V shows the run times of the adaptive serial and parallel

schemes for achieving the quality targets set by Strategy 2. For all runs and all circuits on any number of processors, Strategy 4 manages to achieve significantly higher qualities than either Strategy 1 or Strategy 2.

TABLE IV
RESULTS FOR ADAPTIVE STRATEGY 4 (STRATEGY 1 QUALITIES)

Circuit Name	Number of Cells	$\mu(s)$ SA	Serial SA Time	Time for Parallel SA Strategy 2				
				p=3	p=4	p=5	p=6	p=7
s1196	561	0.675340	75.4	60.31	47.87	47.34	46.25	42.44
s1238	540	0.699469	115.9	96.45	84.21	67.59	63.05	53.79
s1488	667	0.650381	106.6	77.84	70.62	59.92	51.80	43.38
s1494	661	0.647920	139.7	101.1	77.38	76.68	59.68	50.12

TABLE V
RESULTS FOR ADAPTIVE STRATEGY 4 (STRATEGY 2 QUALITIES)

Circuit Name	Number of Cells	$\mu(s)$ SA	Serial SA Time	Time for Parallel SA Strategy 2				
				p=3	p=4	p=5	p=6	p=7
s1196	561	0.630367	37.35	23.71	23.24	21.74	20.57	17.95
s1238	540	0.630573	45.85	33.76	24.52	19.65	23.53	15.03
s1488	667	0.582884	29.59	21.35	18.26	13.36	13.46	12.84
s1494	661	0.591114	46.92	27.78	20.09	20.14	17.68	18.16

As can be seen, both the serial and parallel run times have improved dramatically over Strategy 1, while the runtimes are largely equivalent to those of Strategy 2.

Note however, that the speedup characteristics of Strategy 4 are very similar to those of Strategy 1: for the given quality values, speedup never exceeds 3 (Figure 3).

III. DISCUSSION AND ANALYSIS

The intelligence of SA lies in its cooling-schedule. In AMMC PSA, each independent parallel search chain periodically starts its search from the best available solution at the time. This allows the parallel search to be focused around a recent best solution, which would be the logical place to look for an even better solution. Thus not only does the algorithmic intelligence remain undivided, it is further enhanced using the AMMC approach, allowing the achievement of better solutions in the same or lesser amount of time, as is the case for Strategies 1 and 4.

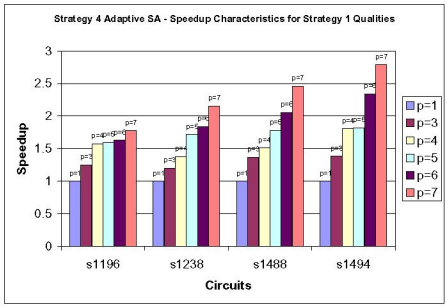


Fig. 3. Speedup Characteristics of Parallel Adaptive SA (Strategy 4) for solution qualities of Strategy 1

As for Strategies 2 and 3, however, we see that although a division of the workload has a positive impact on runtime, there is an adverse impact on achievable quality. This can be understood by looking at how the intelligence of the algorithm is affected by such a division (achieved simply by dividing the cooling-schedule parameter M by the number of processors). Since SA convergence is highly sensitive to the cooling schedule, it is understandable that such a drastic change to one of its parameters would result in lower quality solutions. Division of M reduces the amount of time each processor spends searching for a better solution in the vicinity of a previous good solution, resulting in a less thorough parallel search of the neighboring solution space. Even the proposed enhancement of varying other parameters across other processors, as done in Strategy 3, is insufficient to counteract the impact of divided M.

Strategy 4 parallel SA approach was implemented after a careful study of the impact of varying M on achievable solution quality. The adaptive nature of the cooling schedule allows this technique to achieve high quality results in significantly reduced runtimes, when compared with the original implemented algorithms. However, compared to the serial SA version with a similar Adaptive cooling schedule, the speedup benefits are less significant, in fact more comparable to the difference between Strategy 1 and the original Serial SA - achieving the same quality solution in slightly lesser time.

IV. CONCLUSION

In this paper, we have presented 4 distinct implementations of AMMC PSA. Strategy 1 provides significantly better solution qualities than the serial algorithm, but only modest speedup. Strategies 2 and 3 suffer a quality loss of 10 to 15%, but provide near linear speedups for the achieved qualities. Our best parallel implementation in terms of both solution quality achievable and run time was Strategy 4 - a new implementation of Simulated Annealing utilizing an adaptive cooling schedule. Both the serial and parallel versions of this approach have shown dramatic improvements in achievable quality and runtime over the basic Simulated Annealing implementation, as well as over the first three Strategies. We are currently exploring further methods of enhancing this scheme in an attempt to achieve better speedup characteristics.

ACKNOWLEDGMENT

The authors thank King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia, for support under Project Code COE/CELLPLACE/263.

REFERENCES

[1] S.-Y. Lee and K. G. Lee, "Synchronous and asynchronous parallel simulated annealing with multiple-markov chains," *IEEE Transactions on Parallel and Distributed Systems*, vol. 7, no. 10, pp. 993–1008, October 1996.

[2] J. Chandy, S. Kim, B. Ramkumar, S. Parkes, and P. Bannerjee, "An evaluation of parallel simulated annealing strategies withapplication to standard cell placement," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 4, pp. 398–410, April 1997.

[3] J. A. Khan, S. M. Sait, and M. R. Minhas, "Fuzzy Biasless Simulated Evolution for Multiobjective VLSI Placement ," *In Proceedings of the IEEE Congress on Evolutionary Computation, CEC'2002.*, vol. 2, pp. 1642–1647, 2002.

[4] S. M. Sait and H. Youssef, *VLSI Physical Design Automation: Theory and Practice*. World Scientific, Singapore, 2001.

[5] P. Banerjee, *Parallel Algorithms for VLSI Computer-Aided Design*. Prentice Hall International, 1994.

[6] V.-D. Cung, S. L. Martins, C. C. Riberio, and C. Roucairol, "Strategies for the Parallel Implementation of metaheuristics," *Essays and Surveys in Metaheuristics*, pp. 263–308, Kluwer 2001.

[7] K. Konishi, K. Taki, and K. Kimura, "Temperature parallel simulated annealing algorithm and its evaluation," *Transactions on Information Processing Society of Japan*, vol. 36, no. 4, pp. 797–807, 1995.